System design document for the BrawlBuddies project (SDD)

Contents

Version: 2.0

Date: 2014-05-25

Author: David Gustafsson, Matz Larsson, Lisa Lipkin, Patrik Haar

This version overrides all previous versions.

1 Introduction

1.1 Design goals

The design should be modular and have loose connections so that it would be possible to switch the controller or view without having issues. The design shall make it possible to add on a server-client architecture to the application. The model must be able to add feature with ease. For usability see ***RAD***.

1.2 Definitions, acronyms and abbreviations

Java:	Platform independent programming language
JRE:	The Java Runtime Environment: Additional software to run a Java application.
MVC pattern:	A software architectural pattern that divides the application in three parts: Model, View and Controller. This satisfies the guideline "Separation of concern"
PNG:	An image file format
XML	"Extensible Markup Language". Text markup language.
OGG	A sound file format
TMX	A XML-based file following the standards of Tiled (see references)
I/O	Input/Output information processing between application and the outside world

2 System design

2.1 Overview

The application will use a modified MVC model.

2.1.1 The model functionality

The models functionality will be exposed by the interface IBrawlBuddies. IBrawlBuddies will be the top level interface acting as an entry to other interfaces in the model.

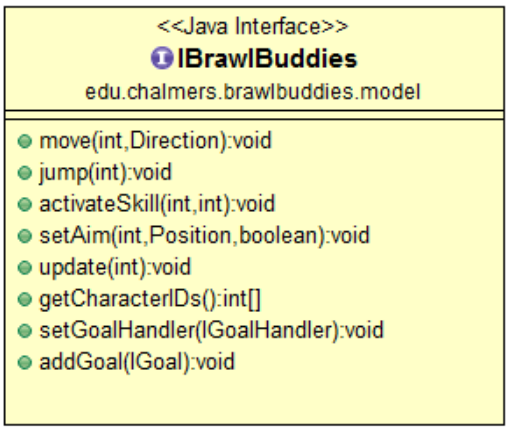

Figure 1: UML diagram of the interface IBrawlBuddies

2.1.2 Application goal

The goals of the game can be set in the menu system with a default of getting the opponent health to zero three times. When the goals of the game is completed a game over event is sent by the model to controller. The goals are handled by the class GoalHandler where different goals can be added in various combinations.

2.1.3 Unique identifiers and look-ups

Every object has a unique identifier. This is used by the model to match player with player controlled character and by effects to detect the player that created them. It is also used by the view for matching objects with their visual representation. The visual representations will be stored in a global accessible data structure.

2.1.4 Event handling

Many events can happen during game play. To avoid creating a web of connections, an “eventbus” is implemented. All essential changes in the model sends an event via the eventbus to inform about the change. The events are sent by so called wrappers, which act like a layer between the model and the eventbus. Wrappers can access the eventbus at any time in a static way.

2.1.5 Resource Documents

To easily add new character with new abilities, characters, skills, projectiles and melee data is stored in modifiable xml documents.

2.1.6 Internal representation of text

The text will be written in English only. No translations will be available.

2.2 Software decomposition

2.2.1 General

The application is composed in following modules

- model - Contains the game logics. Model part of the MVC application
  NOTE: Subpackages of this is only used for easier navigation for developers.

- view - Handles the GUI. View part of the MVC application.

- controller - Handles the input and distributes the work.Controller part of the MVC application

- util - Contains helpful methods for running the application: I/O-handling, logging and supplementary methods to our current libraries.

- eventbus - Contains the EventBus singleton and its help-classes for eventbus communication.

- Constants - Contains the constants used in the game.

- Main - The main class used for starting the application.

2.2.2 Subpackages

2.2.2.1 model - Contains the all classes and packages that is needed to create and run the model part of the application. The package contains all goal logics. This package is also responsible for basic descriptions of location and direction.
NOTE: Subpackages of this package is used for easier navigation for the developers.

2.2.2.1.1 model.world - Contains the objects of the game world such as the characters, projectiles, the static game environment etc. It also contains logic for handling actions and reactions of these objects during runtime, such as collisions and movement.

2.2.2.1.2 model.skills - Contains the framework of the abilities the player can use to interact with the world. The skill framework contains both the parts needed to create skills for the character and the effects the skills apply to characters.

2.2.2.1.3 model.statuseffect- Contains the framework of the effects that can afflict the character. It also handles logic for managing several status effects on a character.

2.2.2.2 view - Contains all the classes and packages that are needed to create and run the view. It also handles connections between game objects and images.

2.2.2.2.1 view.sound - Contains the classes to load, build and play music and sounds.

2.2.2.2.2 view.menu - Contains the visual representation of the game menus.

2.2.2.3 controller - Contains all the classes and packages that is needed to create and run the Controller. Handles input, controls the state of the game and distributes all work.

2.2.2.3.1 controller.input - Handlers for all input to the game

2.2.2.3.2 controller.menu - Handles the input from menus and the navigation.

2.2.2.4 util - Contains helpful classes to run the application. This includes support for I/O, logging and supplementary methods to our current libraries.

2.2.2.5 eventbus - To facilitate the separation of concern an eventbus system is implemented to send events from the model to the view. This system has the further benefit of making it easy to add on more listeners if there was a need to record statistical information about the game i.e. how damage was done in the match.

2.2.3 Decomposition into subsystems

The application subsystems consist of a several utility classes. This includes I/O-support, logging support, classes that contains copy methods for lists and shapes and a shape factory.

2.2.4 Layering

The layering is indicated in the figure below where higher layers are at the top of the figure.

2.2.5 Dependency analysis

The game is dependent on the external library Slick2d. Further dependencies are shown in the figure below

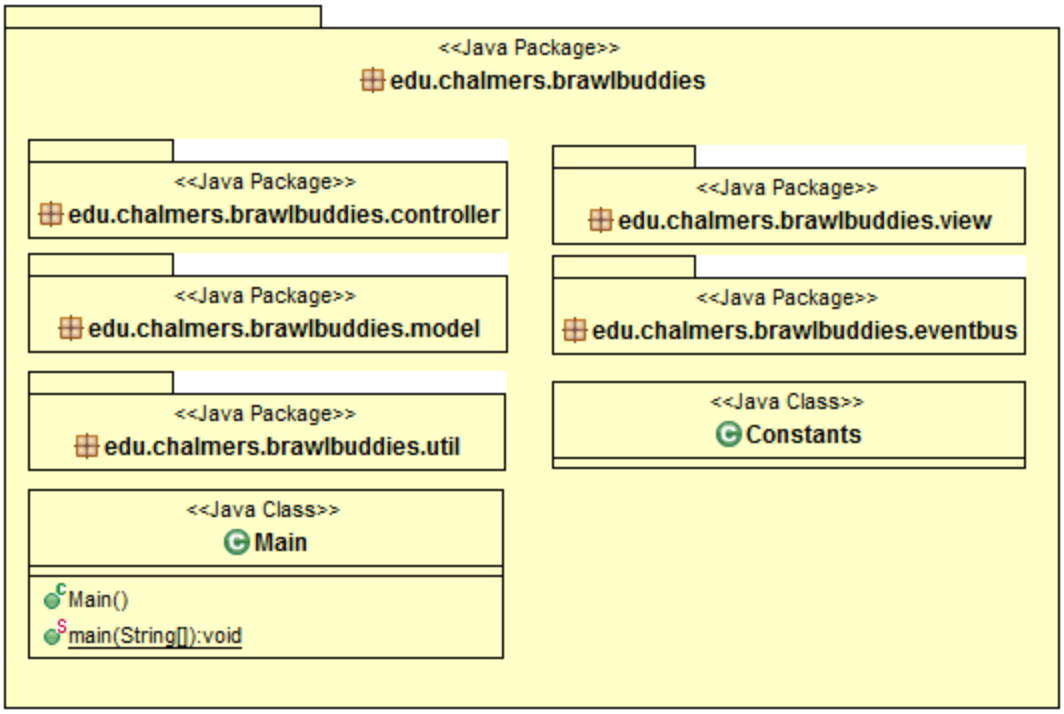

Figure 2: High level design

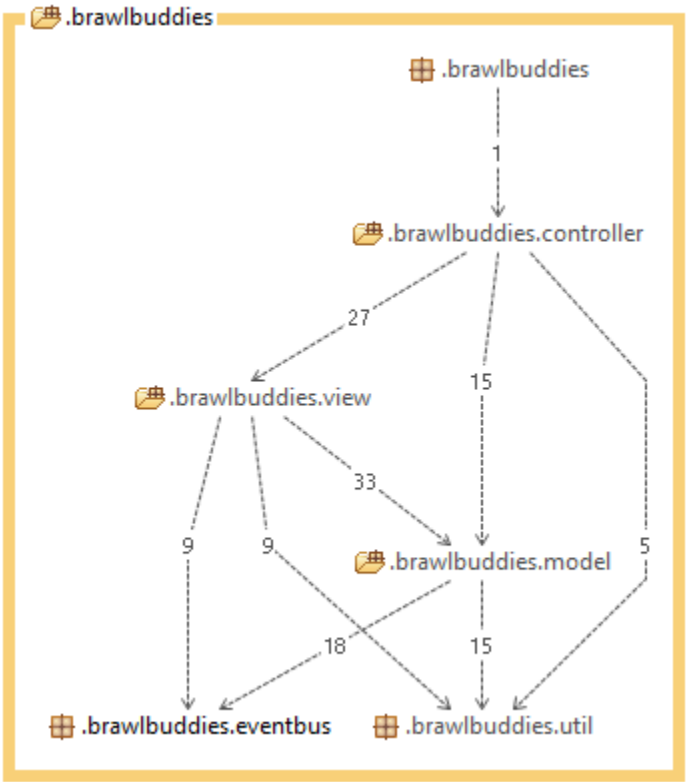

Figure 3: Layering and Dependency analysis

2.3 Concurrency issues

NA .The application will run from a single thread.

2.4 Persistent data management

All persistent data is will be stored in XML (data), TMX (maps), OGG (sound) or PNG (images) files in a resources folder.

2.5 Access control and security

NA

2.6 Boundary conditions

NA. The application should be able to launch as a jar-file and closed from the application or from the desktop manager.

3 References

1. MVC, see http://en.wikipedia.org/wiki/Model-view-controller
2. PNG, see http://en.wikipedia.org/wiki/Portable_Network_Graphics
3. XML, see http://en.wikipedia.org/wiki/XML
4. OGG, see http://en.wikipedia.org/wiki/Ogg
5. TMX, see https://github.com/bjorn/tiled/wiki/TMX-Map-Format